# Optimized GPU-accelerated Monte Carlo program for real-time dose estimation directly using mesh-type computational phantoms[*]

Shuchang Yan,[1,2] Rui Qiu,[1,2,†] Ankang Hu,[1,2] Shuiyin Qu,[1,2] Yang Zhou,[1,2] Ziyi Hu,[1,2] Yanhan Zhou,[1,2] Zhen Wu,[1,2,3] and Junli Li[1,2]

[1]*Department of Engineering Physics, Tsinghua University, Beijing 100084, China*
[2]*Key Laboratory of Particle & Radiation Imaging, Tsinghua University, Ministry of Education, Beijing 100084, China*
[3]*Nuctech Company Limited, Beijing 100084, China*

Mesh-type phantoms represent the latest generation of human computational phantoms, offering high resolution and adjustability advantages for individualized radiation dosimetry. Current dosimetry computation methods, which require conversion to tetrahedral mesh models for efficient Monte Carlo simulations, still do not meet the requirements for real-time dose calculations. Advancements in heterogeneous computing now allow for significant acceleration in mesh-type phantom calculations by utilizing both high-performance hardware and efficient algorithms. This study aims to develop a GPU-accelerated Monte Carlo simulation method that directly utilizes mesh-type phantoms to further enhance the speed of human dose calculations without the need for tetrahedralization. For the boundary representation polygonal models, this study redesigned and implemented the entire procedural flow of the GPU-accelerated Monte Carlo program, developing particle transport methods within the mesh-type model. All triangular facets of the mesh-type model were constructed into a tree-like acceleration structure and the traversal access pattern was optimized. Moreover, this study adopted an event-based transport method, transporting particles step-by-step by particle type, and a bias-based variance reduction technique employing geometric weights was integrated. For typical external irradiation scenarios, dose calculations between Geant4 and our GPU-based program were compared to assess computational accuracy and efficiency. Compared to the benchmark simulations conducted on a single-thread CPU via Geant4, the organ dose discrepancies calculated by the GPU-accelerated program generally remained within a 5% margin, while computational times were reduced by factors ranging from 500 to 50000. To our knowledge, this study is the first to utilize a mesh-type model for GPU-accelerated dose calculation without tetrahedralization. The simulation time has been dramatically reduced from hours to just mere seconds, offering a rapid and precise Monte Carlo method for mesh-type computational phantoms. This development supports real-time dose calculation studies using dynamic mesh-type models, providing a robust Monte Carlo simulation tool.

Keywords: GPU Monte Carlo, Mesh-type phantom, Heterogeneous, Real-time dose

## I. INTRODUCTION

Monte Carlo (MC) simulations employing computational phantoms serve as a crucial method for human dose assessment. The boundary-represented mesh-type models, as the latest generation of computational phantoms, offer the dual benefits of flexible deformability and high resolution, which better represent the dosimetric characteristics of real human body [1–3]. Consequently, mesh-type phantoms have demonstrated significant potential in the fields of radiation therapy, radiation protection, and individual dosimetry, where more accurate models are essential for obtaining precise individual doses [4–7].

However, employing mesh models directly in MC simulations for dose calculations introduces challenges [8]. In MC simulations, accurately defining the relationships between particles and their surrounding geometrical structures is essential [9, 10]. Direct use of mesh models for these computations notably reduces computational speed, with research indicating that such use increases computational time by 70 to 150 times compared to voxel models [11]. In response, a novel method involving tetrahedral decomposition [12, 13] has been developed to expedite computations for mesh phantoms [14, 15], and the resulting tetrahedral models also support posture adjustments [16]. While this tetrahedralization simplifies the determination of geometric relationships during particle transport, the subdivision of the mesh into tetrahedra substantially increases the internal complexity, which impedes further acceleration of computation speeds [12]. Thus, in fields like clinical radiotherapy, nuclear medicine, and accident dose reconstruction, where strict time constraints are crucial, the computational time of mesh-type models often exceeds acceptable limits, restricting their applicability in these critical areas [17, 18].

In recent years, the rapid development of GPU hardware and continual optimization of ray-tracing software algorithms have made it feasible to directly employ boundary-represented models for dose calculations [19–22]. GPUs, with their superior floating-point computational capabilities and increased thread count, are better suited for large-scale particle simulations compared to CPUs [23, 24]. Consequently, numerous GPU-accelerated programs for photon and photon-electron coupled transport have been developed, achieving significant acceleration [18, 25–29]. Our group has previously developed the first GPU-accelerated MC transport program based on tetrahedral phantoms. However, due to issues with GPU thread divergence [30] and the increased internal complexity caused by tetrahedralization [12], the com-

[*] Supported by the National Natural Science Foundation of China (Grant No. U2167209 and Grant No. 12375312)

[†] Corresponding author, qiurui@tsinghua.edu.cn

putational time of the tetrahedral GPU-accelerated MC program remains at the level of tens of seconds to minutes. Once the direct computation with mesh-type phantoms is achieved, bypassing the tetrahedral segmentation step and addressing the slow particle-to-geometry positioning, there is potential for further improvements in computational efficiency. To the best of our knowledge, current popular GPU MC programs have not yet implemented GPU acceleration directly based on mesh-type models. The principal challenge lies in the complexity of implementing rapid particle transport within mesh-type models on GPUs [31]. Additionally, GPU-based MC programs often face significant thread divergence issues due to the considerable variability among different particle transport processes [30, 32]. Furthermore, smaller organs tend to present larger statistical errors in dose results because of the lower probability of particle interactions [33], necessitating an increase in the number of simulation particles to achieve more accurate dosimetry, which significantly extends the simulation time.

To further enhance the computation speed for dose calculation in mesh-type phantoms, this study implemented GPU-based Monte Carlo simulation program directly utilizing boundary-represented mesh models. This involved redesigning the entire procedural flow of particle transport. All triangular facets of the mesh model were organized into a tree-like acceleration structure and the traversal access pattern was optimized, significantly reducing the complexity of geometric localization and intersection calculations during particle transport. Furthermore, this study adopted an event-based transport method, conducting multiple simulations in which particles of the same type were transported for a single step during each simulation, rather than relying on a single thread to transport one particle until termination, which greatly reduced thread divergence and improved hardware utilization. The introduction of multi-GPU parallel processing further accelerated the computation speed. Additionally, we employed a biasing sampling technique based on geometric weights for variance reduction, significantly reducing statistical errors in smaller organs and decreasing the number of simulated particles. Rigorous validation of the program demonstrated precise computational outcomes and substantial acceleration, effectively addressing the challenges associated with enhancing dose calculation speed for mesh-type phantoms.

## II. MATERIAL AND METHODS

This study presents the development of a program that directly utilizes a mesh-type model for GPU-accelerated MC simulations. Various optimization techniques, including event-based transport, multi-GPU parallelism, and variance reduction methods, were then implemented to further enhance the computational speed. Finally, the accuracy of the GPU program was validated, and the effectiveness of the optimization methods in accelerating the simulations was assessed.

### A. GPU Monte Carlo program for mesh-type phantom

#### *1. Constructing flat acceleration structure*

In MC simulations, determining both the physical and geometric step sizes requires the material cross-section information at the particle's location, as well as the distance to the boundary along the particle's trajectory [9, 10, 34]. For mesh-type phantoms, it is necessary to traverse all triangular facets, to perform particle localization and intersection operations. The repetitive traversal operation, which must be performed for each particle transport step, is time-consuming. To enhance traversal efficiency, the implementation of acceleration structures becomes crucial [35–37]. These structures systematically organize data into layers, significantly reducing the number of searches and enhancing query efficiency. Given this context, we construct a tree-based acceleration structure for all triangular facets of the mesh-type phantom, to substantially reduce the time complexity of data traversal. Considering both time complexity and the need for dynamic updates beneficial for phantom adjustments [38], we choose to implement a Bounding Volume Hierarchy (BVH) tree, which is widely used in ray tracing and animation [39].

The constructing process of BVH tree begins with calculating a bounding box for all triangles, which forms the parent node [39]. These triangles are then divided into two groups based on a specific pattern, such as an average division by quantity. The bounding box calculation and division process continue for the two child nodes until the number of triangles in a subdivided child node falls below a specified threshold, at which point it becomes a leaf node and stores the information of the included triangles, as illustrated in figure 1. An effective partitioning strategy is essential. When geometric objects in a scene are unevenly distributed, traditional partitioning methods may yield an unbalanced tree structure, which can reduce traversal efficiency. To optimize this process, the Surface Area Heuristic (SAH) is typically employed [39, 40]. The SAH evaluates the cost of each partition by calculating the surface area of the resulting child bounding boxes. This method explores various partitioning schemes and selects the one with the lowest cost for implementation. Such optimization can significantly reduce unnecessary intersection tests between rays and bounding boxes during ray tracing, thereby enhancing overall intersection efficiency.

In mesh-type phantoms, the presence of numerous triangular facets results in a deeply nested and complex acceleration structure. This complexity presents significant challenges for GPU computing, which is limited by stack depth [31]. Such constraints restrict the number of recursive calls and the depth of data structures that can be processed, adversely affecting the performance of recursive algorithms traversing these structures. To overcome this limitation, the flattening of acceleration structure becomes essential. This process involves converting the tree structure into a linear format for efficient processing on GPUs [41]. Our implemented "tree flattening" method is illustrated in figure 1. We record the indices of child nodes separately in the node sequence rather than creating new child nodes directly within each node. This strategy

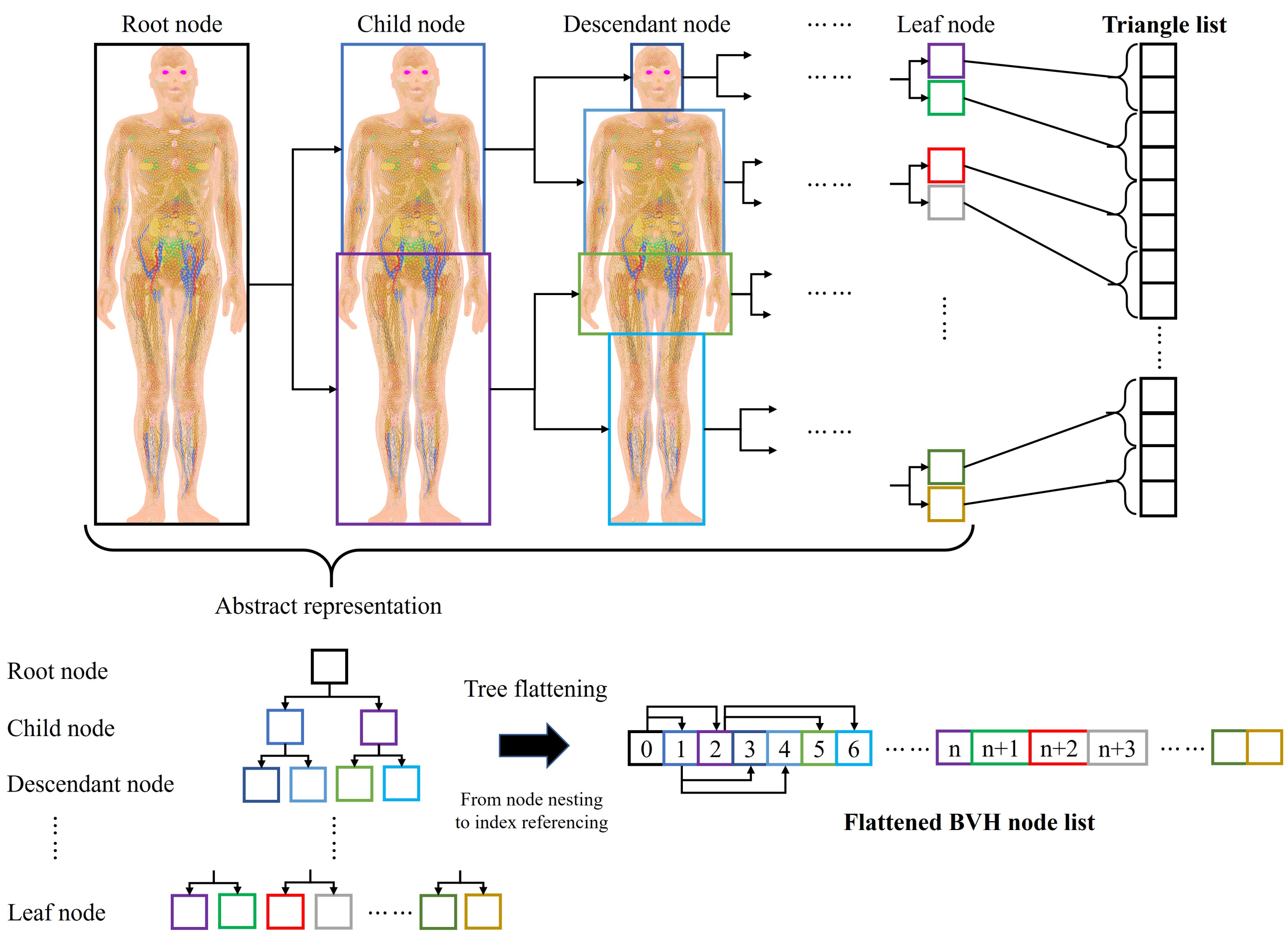


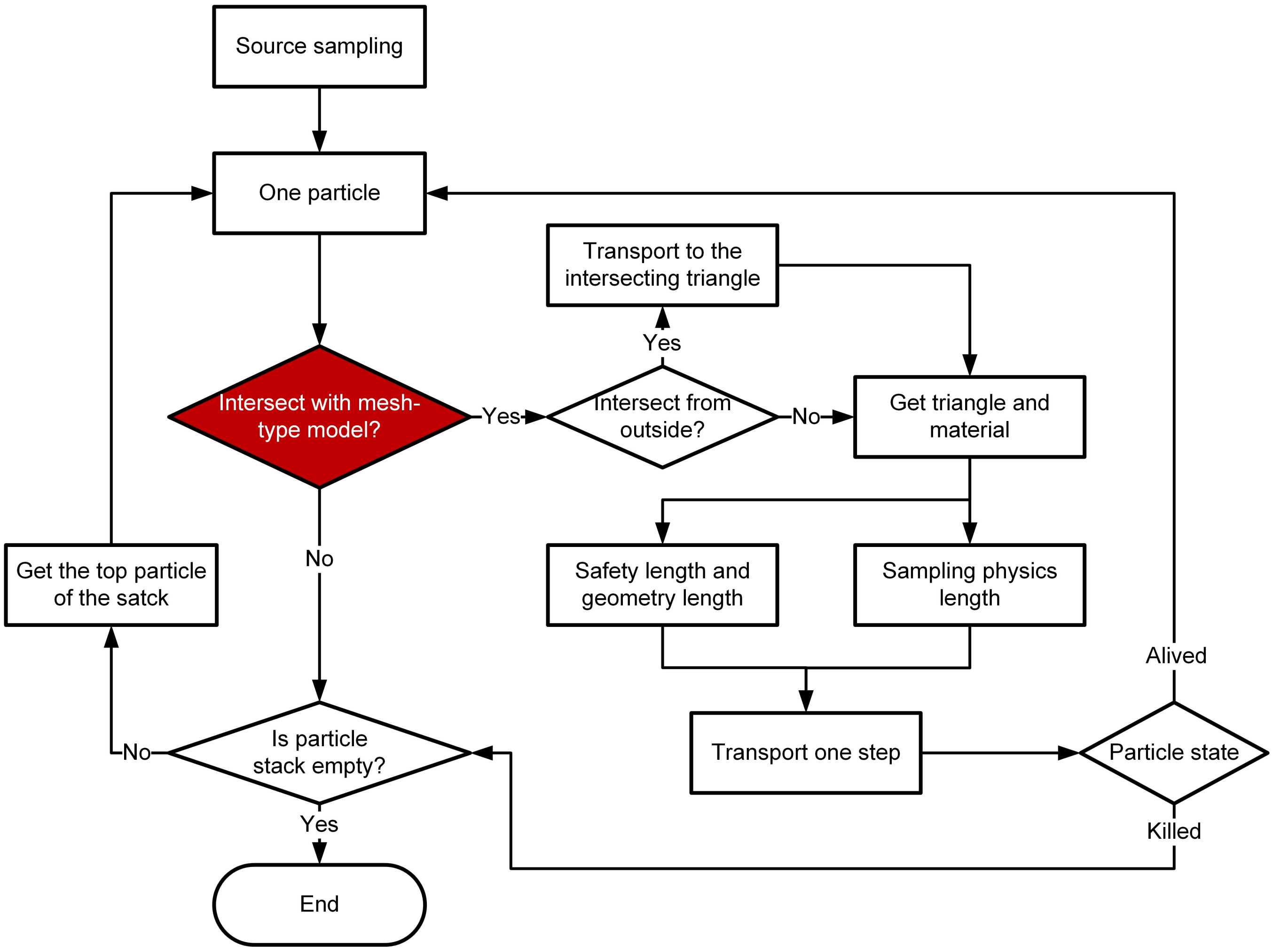
Fig. 1. (Color online) Flowchart for constructing flat acceleration structure of the ICRP mesh-type phantom. The upper part of the diagram shows the step-by-step division of the surface model into bounding boxes to build a tree structure. This division continues until the number of triangles within a bounding box falls below a threshold, at which point the division stops, and the node becomes a leaf node, storing the indices of the encompassed triangles. The lower part of the diagram provides an abstract description, followed by tree flattening, where child nodes are referenced in the node list by their indices instead of node nesting.

produces two sequences: one containing a list of BVH nodes that sequentially stores information for each node, and another comprising a list of triangular facets that sequentially records the geometric and material information associated with the leaf nodes. The flattening of the BVH tree structure circumvents the limited stack depth of GPUs by avoiding nested configurations. This architectural adjustment ensures that more complex phantom acceleration structures can be efficiently processed on GPUs.

## *2. Transport in the acceleration structure composed of numerous triangle*

The particle transport process within the mesh model is depicted in the figure 2. Particles are generated through sampling on the GPU and subsequently transported in a stepwise manner until termination [9]. As previously noted, localization and intersection operations represent the most critical and time-consuming aspects of particle transport. Localization is necessary to obtain the material information of the geometry where the particle resides, allowing for energy interpolation to derive the reaction cross-sections and sample the physical step length. The intersection operation determines the distance to the nearest geometric boundary in the direction of particle motion, thereby determining the geometric step length. By comparing the physical step lengths associated with various reactions to the geometric step length for transport, the minimum value is selected to ascertain whether a reaction occurs or the particle is transported to the boundary [9, 42]. This process is repeated until the particle and its secondary particles are terminated.

Determining the relationship between particles position and mesh structures typically involves emitting a virtual ray from the particle, often aligned with the coordinate axes, as illustrated in figure 3(a). The number of intersections and



Source sampling
One particle
Transport to the intersecting triangle
Yes
Intersect with mesh-type model?
Yes
Intersect from outside?
No
Get triangle and material
No
Get the top particle of the satck
Safety length and geometry length
Sampling physics length
Alived
Transport one step
No
Is particle stack empty?
Particle state
Yes
Killed
End

Fig. 2. (Color online) Particle transport process within the mesh-type phantom composed of numerous triangle.

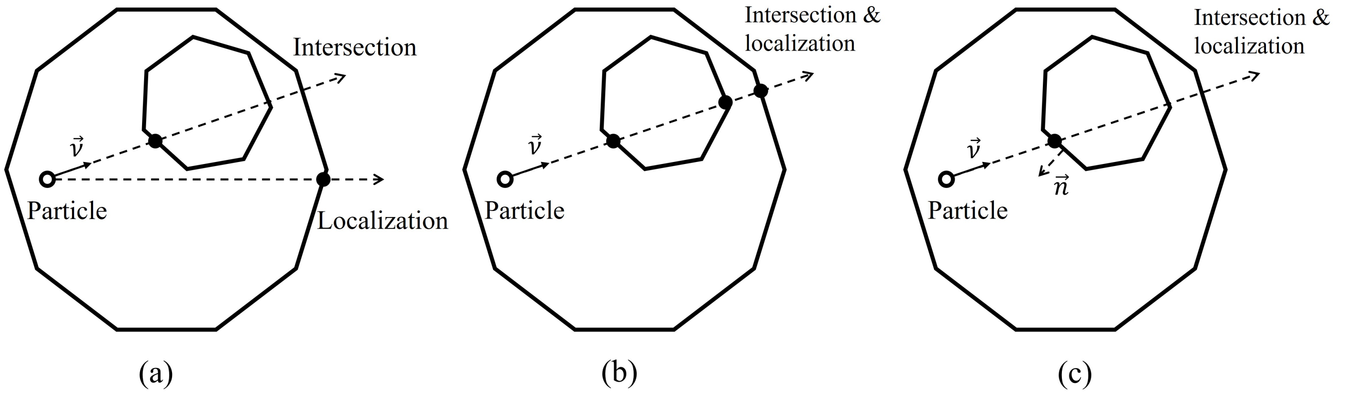


Fig. 3. (Color online) The optimization of particle positioning and intersection operations within the mesh-type phantom. Subfigure (a) illustrates the separate execution of particle positioning and intersection operations. Subfigure (b) demonstrates that the determination of particle positioning and intersection is achieved through the particle direction and geometric intersections. Subfigure (c) shows the evaluation of particle position based on the normal information of the nearest intersecting triangle and the particle direction.

the corresponding distances are used to ascertain the current geometry and material of the particle. Generally, the geometry with an odd number of intersections that is closest to the particle is selected. Calculating the geometric step length requires determining the distance to the nearest grid boundary along the particle's direction of motion, which also necessitates intersection operations. By integrating these two processes, both particle localization and intersection detection can be accomplished simultaneously, as shown in figure 3(b). To further enhance the efficiency of these procedures, we preassign additional information to the triangular facets of the mesh model. Specifically, we add the normal vector for each triangular facet, indicating the outward direction from the interior of the mesh geometry, and specify the materials associated with the outward and inward normal. By identify the nearest intersection point and intersecting triangle, we can determine the material in which the particle is currently located by assessing the relationship between the particle's motion direction and the triangle's normal direction, as depicted in figure 3(c). This approach eliminates the need to count intersections and assess the number of intersection points with the same geometry, thereby reducing computational time.

The preceding discussion focused on the transport of particles after their intersection; however, it did not detail the methods for quickly intersecting a large number of triangles based on particle position and directional information. This is where the acceleration structure we previously constructed becomes essential. We will establish an empty stack of intersection nodes, placing the first node of the flat acceleration structure node sequence at the top of this stack. Next, we will fetch the top node from the stack. If this node is not a leaf node, we will perform an intersection operation between the particle's direction of motion and the two child bounding boxes of this node. If an intersection occurs, the corresponding child nodes of the intersecting bounding boxes will be added to the top of the stack. If the fetched node is a leaf node, we will traverse the few triangular facets within it to perform ray-triangle intersections, recording information about any intersecting triangles. This process continues until the stack is empty, at which point we select the shortest intersection distance as the geometric step length and determine the intersecting triangle. Subsequently, we will use the material information associated with the inward and outward normal to identify the material at the particle's position. Pseudocode illustrating this more intuitively is shown in the algorithm 1.

**Algorithm 1:** Particle intersection and localization within acceleration structure

```
Input: Particle position, particle direction, and
       acceleration structure
Output: Result of intersection and localization
1  Initialize an empty stack of nodes;
2  Push the root node onto the stack;
3  $min\_distance = \infty$;
4  while stack is not empty do
5      $node$ = Pop top node from stack;
6      if node is a leaf node then
7          for each triangle in node.triangles do
8              Check if particle intersects with the
                triangle;
9              if intersection distance < min_distance
                then
10                 $min\_distance$ = intersection distance;
11                 Update intersecting triangle;
12             end
13         end
14     else
15         if particle intersects with the bounding box of
            left child node then
16             Push left child node onto the stack;
17         end
18         if particle intersects with the bounding box of
            right child node then
19             Push right child node onto the stack;
20         end
21     end
22 end
23 if intersection exists then
24     $triangle$ = Get intersecting triangle;
25     $material =$
        dot product$(triangle.normal, particle.direction) >$
        $0\ ?\ triangle.inMat : triangle.outMat$;
26     $geometric\_step\_length = min\_distance$;
27 else
28     Kill particle;
29 end
```

## 3. Program implementation framework

Given that dosimetry simulations in the human body involve the physical processes of coupled photon-electron transport, the GPU MC program developed in this study provides a comprehensive simulation of these coupled transport processes, utilizing the same physical models and cross-sectional data previously researched by our team [43]. Furthermore, the foundational framework and initialization procedures remain consistent, requiring only the integration of the reading mesh models and construction processes for acceleration structure. On the GPU side, particle transport employs the transport method described in the previous section. The workflow of the GPU MC program for photon and electron transport using the mesh model is illustrated in figure 4.

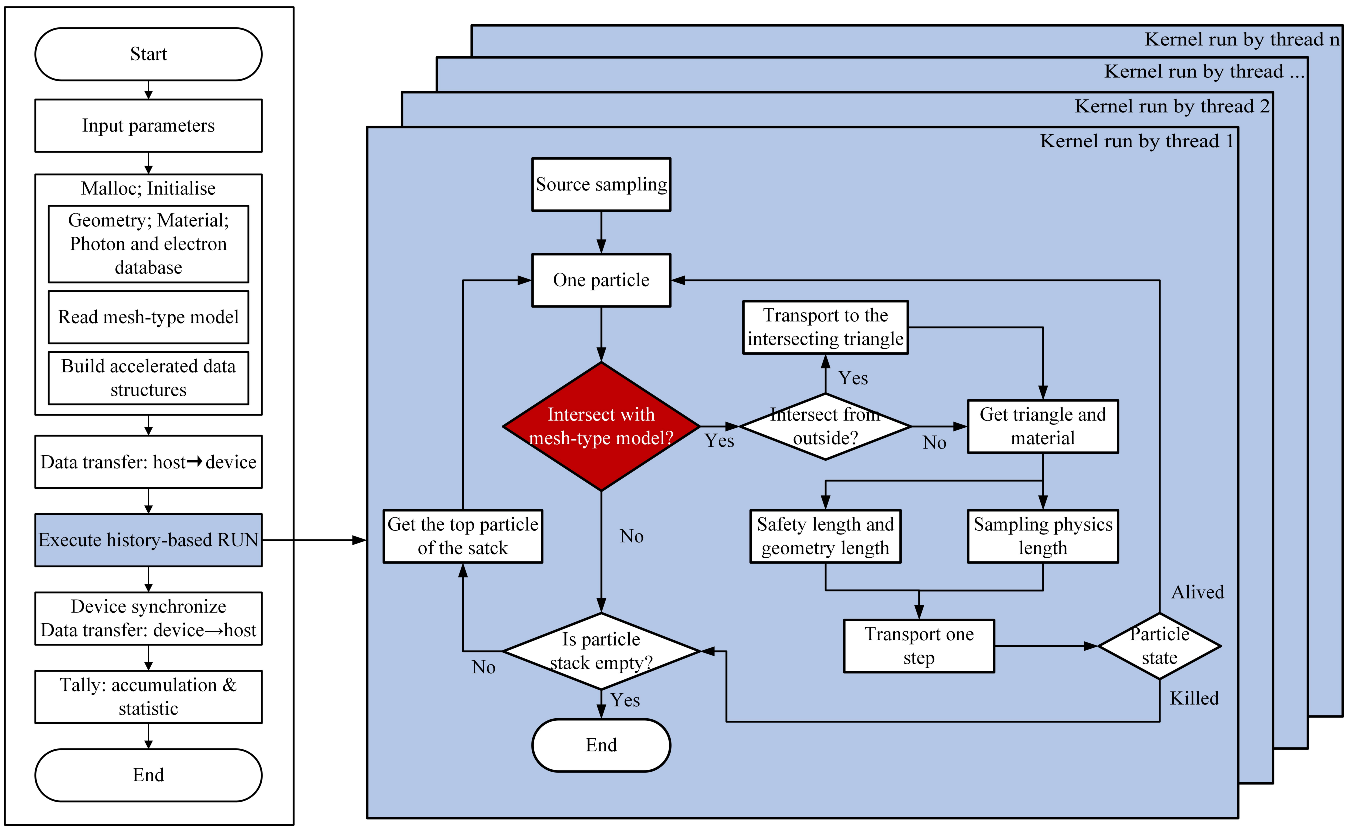

Start
Input parameters
Malloc; Initialise
Geometry; Material; Photon and electron database
Read mesh-type model
Build accelerated data structures
Data transfer: host→device
Execute history-based RUN
Device synchronize Data transfer: device→host
Tally: accumulation & statistic
End
Kernel run by thread n
Kernel run by thread ...
Kernel run by thread 2
Kernel run by thread 1
Source sampling
One particle
Transport to the intersecting triangle
Yes
Intersect with mesh-type model?
Yes
Intersect from outside?
No
Get triangle and material
Get the top particle of the satck
No
Safety length and geometry length
Sampling physics length
Alived
Is particle stack empty?
No
Transport one step
Particle state
Yes
Killed
End

Fig. 4. (Color online) The history-based transport workflow of the GPU MC program using the mesh model. Modules within the blue areas indicate operations performed on the GPU.

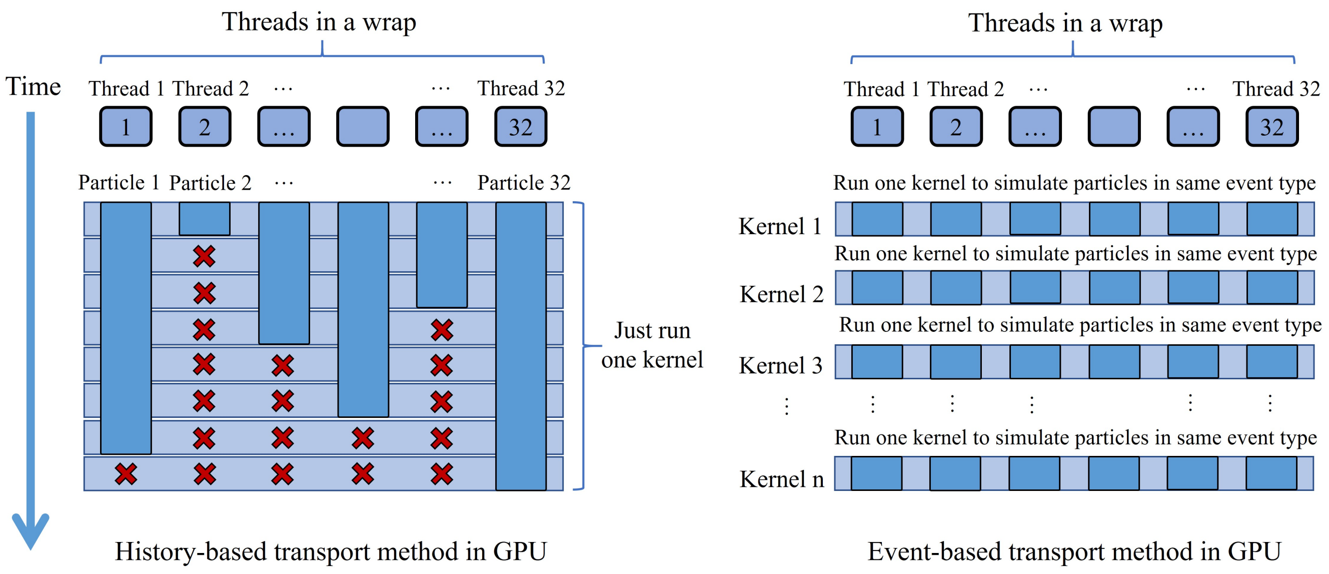
A history-based transport method is utilized, in which each GPU thread simulates a single particle from its generation to termination [9, 32]. Thus far, we have successfully developed a GPU MC program for dose calculation that directly utilizes the mesh model. The subsequent sections will focus on optimizing and enhancing this program.

## B. Event-based transport method

### 1. *Solving thread divergence issues*

GPUs contain numerous threads, which are organized into groups of 32, known as "warps", for efficient management and scheduling. Within a warp, all 32 threads execute instructions in a lock-step manner, meaning that any instruction to be executed by any thread in the warp must be performed simultaneously [32, 44]. When a branching instruction is encountered, the entire warp executes the whole branches taken by any thread. In such cases, if different threads within the warp follow distinct code branches, thread divergence occurs. Although divergence does not affect the correctness of individual thread computations, it significantly impacts code performance. This phenomenon is particularly prevalent in GPU MC simulations. Traditional MC particle transport algorithms typically rely on historical transport methods, akin to the GPU MC program implemented in the previous section, where each thread simulates the transport of a single particle until its lifecycle concludes, as illustrated in figure 4. However, during the simulation, the transport path lengths and times of different particles can vary significantly [32, 45]. For instance, some particles may be "killed" after one single step, while others may require multiple transport steps, as shown in figure 5. Consequently, substantial code branching often occurs when different threads within the same warp handle particles with vastly different transport characteristics, leading to severe thread divergence.

To address this issue, relevant studies have proposed an event-based transport method [32, 46, 47], which divides particle processes into distinct event types. Each GPU batch processes only one type of event at a time, and then cycles through them. This approach ensures that all GPU threads execute the same type of event simultaneously, such as performing a single step of photon geometrical transport. Once the transport is completed, the initial or generated secondary particles are reallocated to sort and consolidate the surviving particles, followed by a subsequent execution. As illustrated in figure 5, this method significantly reduces the divergence of GPU threads, thereby improving hardware utilization and computational efficiency.



Threads in a wrap
Time
Thread 1 Thread 2 ··· ··· Thread 32
1
2
32
Particle 1 Particle 2 ··· ··· Particle 32
Just run one kernel
History-based transport method in GPU
Kernel 1
Kernel 2
Kernel 3
Kernel n
Run one kernel to simulate particles in same event type
Event-based transport method in GPU

Fig. 5. (Color online) Comparison of history-based and event-based transport methods on the GPU. The deep blue bars represent the duration of particle traversal. By categorizing event types and executing the same events through multiple kernel calls, the variability in execution across different threads is significantly reduced, thereby minimizing thread divergence.

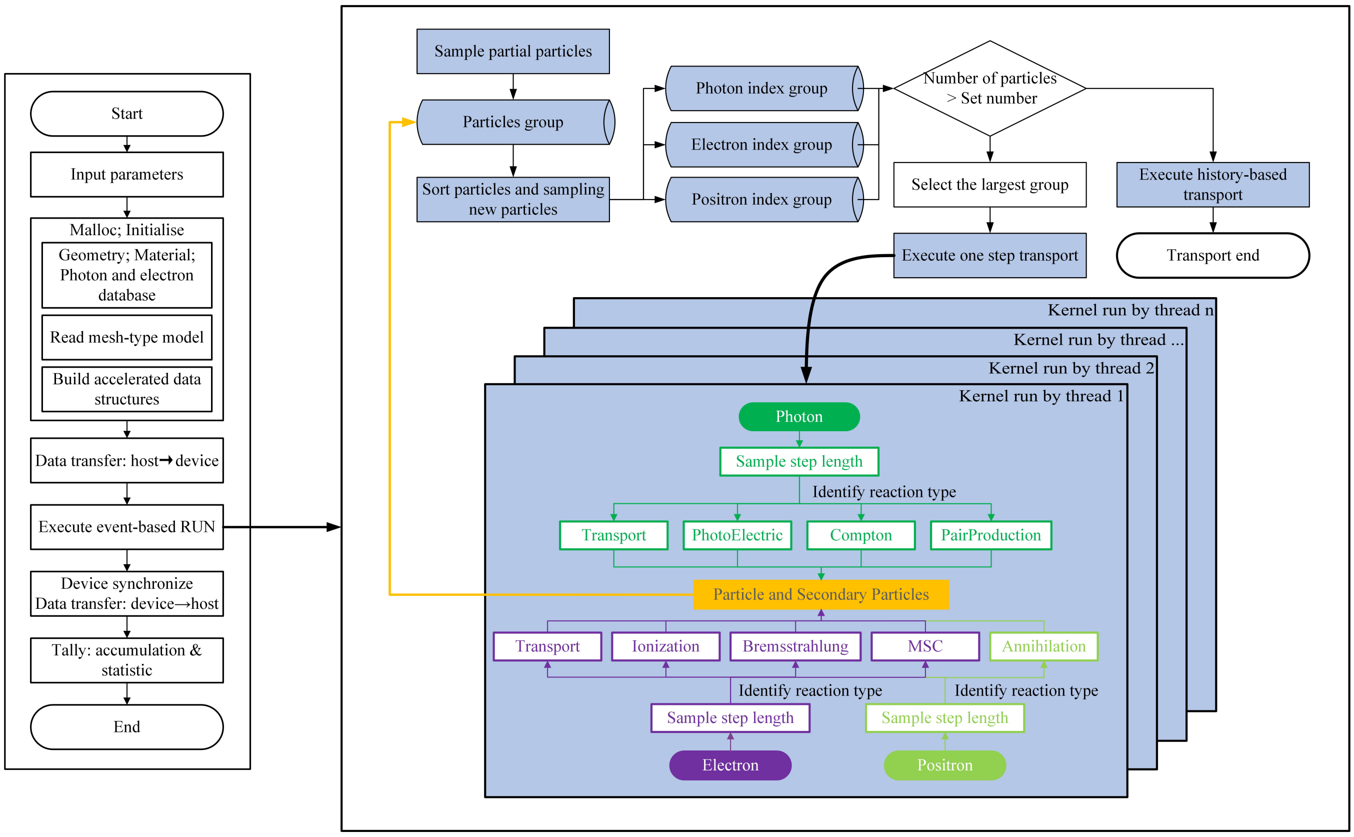


Fig. 6. (Color online) The implementation process of event-based transport method in the GPU MC program for mesh-type phantom. Modules within the blue areas indicate operations performed on the GPU.

### 2. *Implementation method in the GPU program*

Building upon the GPU Monte Carlo program developed in the previous section using the mesh-type model, we extend its functionality to implement an event-based transport method. The implementation process is illustrated in the figure 6. We modify the previous approach of looping through the transport steps of a single particle by removing the loop for particle transport within the GPU kernel function and replacing it with a stepwise transport kernel method. To facilitate classification by particle type, we prepare an event particle library in advance, which stores the particles after each batch transport step. Furthermore, we establish multiple sequences for the different events to record particle indices in the event particle library. The division of particle transport processes for photons, electrons, and positrons allows us to invoke the GPU MC program to transport the particle type with the largest count at each step, as the reaction sampling for the same particle type is generally consistent, resulting in minimal thread divergence. Upon completion of one transport step, we categorize the current particles (if still alive) and the generated secondary particles according to their types and repeatedly invoke the GPU transport program until the number of particles in the event particle library to be simulated falls below a predetermined threshold. The event-based transport method introduces additional operations compared to the history-based approach, such as multiple GPU kernel function calls and the need for particle sorting and data synchronization, which incurs extra time costs. Consequently, when the particle count is low, continuing with the aforementioned event-based transport may reduce efficiency. Therefore, we opt to transport the remaining particles below a predetermined threshold directly using the history-based method until they are terminated.

## C. Multi-GPU parallel optimization

Due to the limitations of single GPU simulations constrained by the number of threads, we considered utilizing multiple GPUs to overcome the current acceleration limits through an increase in hardware. In the previous single GPU workflow, the CPU reads and initializes data, which is then transferred to the device, followed by the GPU executing the simulation computations, and finally, the results are returned to the host for post-processing. Simply replicating this process via multi-threading on the CPU, where each thread independently executes the full workflow using different GPUs, would result in redundant initializations. This redundancy is inefficient due to the substantial size of the phantom data and physical model cross-sections, which can significantly increase processing time and memory usage, potentially exceeding computer memory limits if many GPUs are employed. Moreover, each new simulation iteration, such as updating source term information, requires the complete re-execution of this workflow, leading to considerable delays.

To achieve more efficient multi-GPU parallelism, we transformed the executable simulation program into a library named “Simulation”, followed by a restructuring of the code architecture into independently executable modules. By loading this library in the main program and creating an instance of the simulation class, different functions can be invoked through this object to operate the modules separately. This approach significantly reduces the redundancy of initialization and data transfer processes. The updated code framework is depicted in the figure 7(a). The initialization module is embedded in the constructor of the simulation class. This module is invoked automatically when a new class object is created and is responsible for initializing the physical and geometrical models. This process is executed only once and the data is subsequently transferred to the corresponding GPU. The update module can be executed independently before each simulation to modify source information and update geometric data. This module involves a smaller data volume and also requires transferring the updated data to each GPU. The final component, the GPU particle transport module, utilizes multi-threading to start particle simulation on corresponding GPUs. Considering the significant computational power discrepancies among different GPUs, a load balancing mechanism that assigns simulation particle counts based on the number of CUDA cores in each GPU has been incorporated in this module, as shown in figure 7(b).

## D. Implementation of variance reduction in the GPU MC program

In human organ dose simulations, results are considered reliable when statistical errors are relatively small [2]. However, due to significant variations in the size, position, and shape of different organs, statistical errors also vary considerably. Typically, smaller organs are less likely to be reached by particles, making these interactions rare and requiring a large number of simulated particles for accurate results [33]. In contrast, larger organs do not require such extensive particle simulations. By employing variance reduction techniques that increase particle transport in smaller organs, the total number of particles required for simulations can be significantly reduced, enhancing computational efficiency.

Biasing variance reduction techniques are widely used in mainstream MC simulation programs to decrease computational variance by artificially adjusting particle weights and quantities. Among these, the region importance biasing method is particularly prevalent [48]. The central principle of this method is to assign higher importance to regions of interest and lower importance to less critical areas. Modifications of particle numbers and weights can be achieved through strategies such as surface splitting and Russian roulette based on the importance assigned to these regions. Theoretically, optimal variance reduction can be achieved by precisely setting the bias parameters. Given the fixed structure of human models and the consistent size of internal organs, employing regional importance sampling is appropriate for addressing the disproportionate consumption of particles in obtaining accurate smaller organ dose.

Based on the foundation of this work’s GPU MC simulation program, the process to implement variance reduction techniques through regional importance method is depicted in the figure 8. This process involves assigning importance parameters to both the geometries and the particles. As particles traverse geometric boundaries, they undergo transformations like surface splitting and roulette, both of which alter the particle’s importance depending on the relative im-

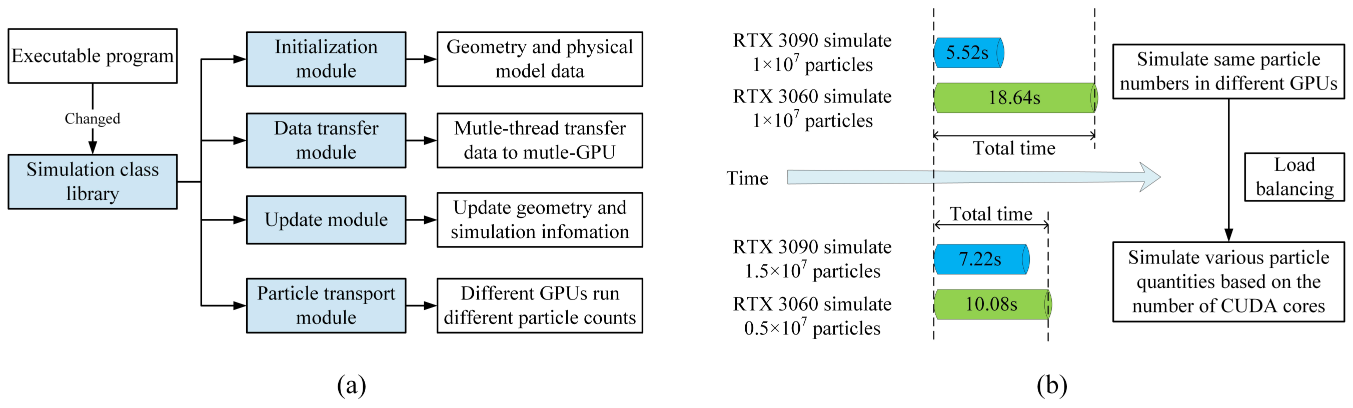

Executable program
Changed
Simulation class library
Initialization module
Data transfer module
Update module
Particle transport module
Geometry and physical model data
Mutle-thread transfer data to mutle-GPU
Update geometry and simulation infomation
Different GPUs run different particle counts
(a)
RTX 3090 simulate 1×10^7 particles
5.52s
RTX 3060 simulate 1×10^7 particles
18.64s
Total time
Time
Total time
RTX 3090 simulate 1.5×10^7 particles
7.22s
RTX 3060 simulate 0.5×10^7 particles
10.08s
Simulate same particle numbers in different GPUs
Load balancing
Simulate various particle quantities based on the number of CUDA cores
(b)

Fig. 7. (Color online) Schematic of Multi-GPU parallel optimization. Subfigure (a) shows the basic modules of the simulation class. Subfigure (b) illustrates effect of load balancing; the upper part displays the performance when different GPUs run the same number of particles, while the lower part shows performance when the number of particles is adjusted according to each GPU's capabilities.

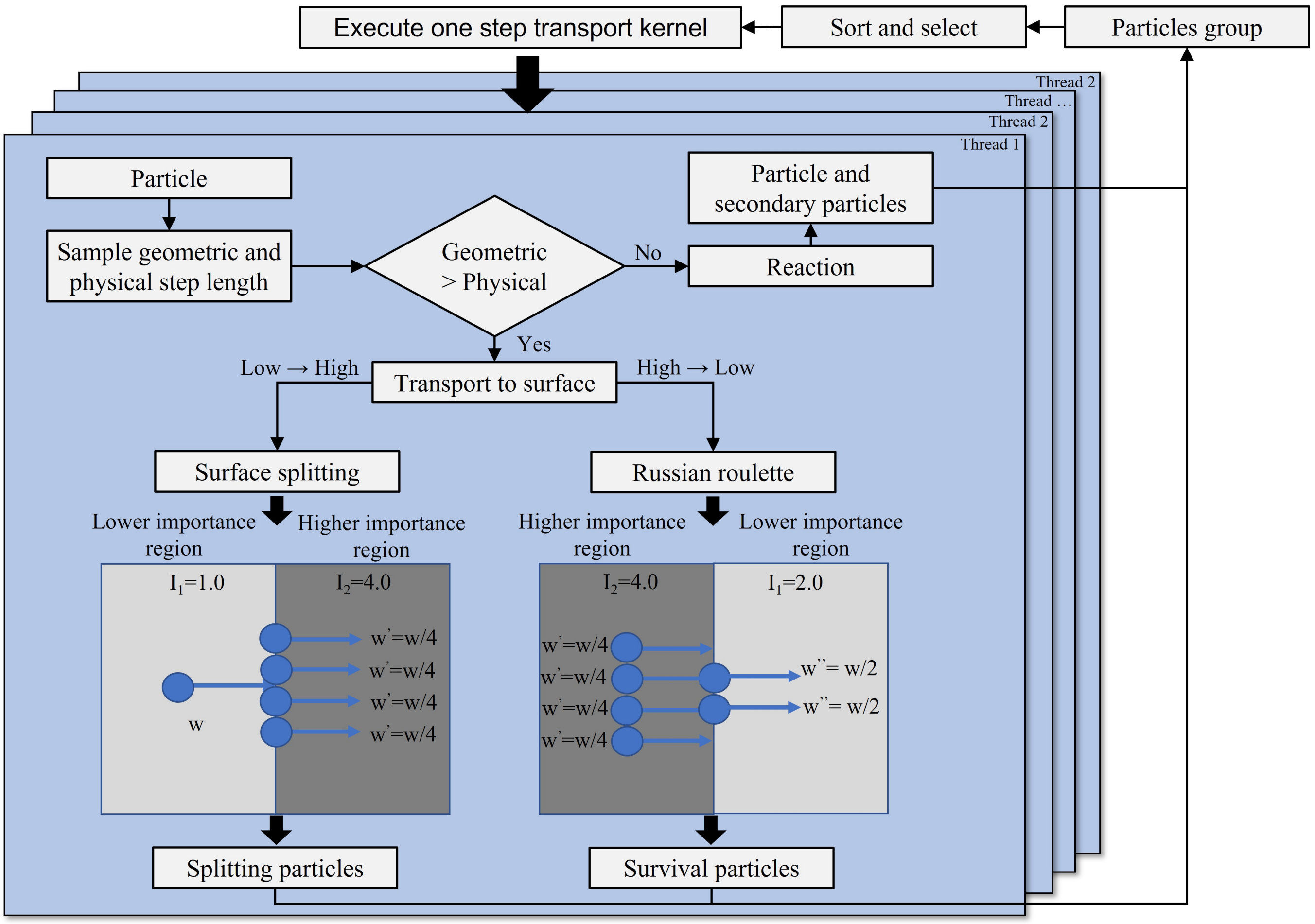


Fig. 8. (Color online) The implementation of variance reduction techniques using regional importance method in the event-based GPU MC program.The areas shaded in blue indicate operations executed on the GPU, while the operations outside the blue areas are performed on the CPU.

portance of the current and subsequent geometries. Surface splitting generates multiple identical particles at the same location, which are then included in the particle sequence for simulation. Roulette involves a probabilistic termination of particles, wherein a particle is eliminated if a randomly drawn number exceeds the ratio of the importance values between the upcoming and current geometries. In the tally module, energy deposition must be weighted based on the current particle's importance.

Region importance sampling, compared to other variance reduction techniques, offers an intuitive principle and straightforward implementation. However, the key lies in the rational design of importance or weight. To this end, simulations based on actual irradiation scenarios can be conducted in advance, with iterative adjustments to region importance for achieving lower statistical errors of organ doses with fewer particles [49]. Variance calculations of our GPU program are conducted using batch-based method [17, 50, 51].

### E. Dose calculations and efficiency assessments

To assess the accuracy and acceleration efficiency of our GPU-based program, we conducted simulations on a standard external irradiation scenario (anteroposterior, AP) using the ICRP's Mesh-type Reference Computational Phantoms (MRCPs) [2]. The male phantom of MRCPs was subjected to unidirectional and parallel photon and electron beams emitted from a planar source, with energies 10 MeV.

For benchmarking, we employed Geant4 simulations on a CPU platform, adapting the source code from the supplemental material of the 145th publication for this AP irradiation scenario [2]. Notably, considering the slow dose calculation speed with the mesh model on CPU, we employed the tetrahedral model for simulation [8, 11]. Our computational setup included Geant4 version 10.04, utilizing the Livermore physics model with secondary electron and photon energy thresholds of 0.2 MeV and 0.002 MeV, respectively. The CPU hardware specifications included an Intel(R) Xeon(R) CPU E5-2660 v4 @ 2.00GHz with 64 GB RAM.

For the GPU program developed by this study, the mesh-type phantom, instead of the tetrahedron model, was directly employed for the same irradiation scenario simulation. To further assess GPU acceleration, we conducted tests to compare computation results and times both with and without event-based transport, variance reduction techniques, and multi-GPU configurations. These simulations were performed on an NVIDIA GeForce RTX 4090 GPU, which boasts 24 GB of video random access memory (VRAM) and operates on Compute Unified Device Architecture (CUDA) version 12.2. The secondary electron and photon energy thresholds and the irradiation scenario were consistent with those of the benchmark.

In summary, we simulated five configurations: (1) CPU simulation $10^8$ particles (benchmark), (2) GPU simulation $10^8$ particles with history-based transport method, (3) GPU simulation $10^8$ particles with event-based transport method, (4) GPU simulation 107 particles with event-based transport and variance reduction, and (5) GPU simulation $10^7$ particles with event-based transport method, variance reduction and multi-GPU mode. The objective was to evaluate the dose calculation accuracy and the acceleration efficiency of the GPU program.

## III. RESULTS AND DISCUSSION

### A. Comparison of calculated dose values

Figure 9 illustrates the dose results from a CPU benchmark and four GPU simulation methodologies incorporating various optimization techniques. Due to the slower performance of CPU calculations with mesh-type models compared to tetrahedral meshes, the CPU benchmark in this study employs a tetrahedral model for comparison. In contrast, the GPU simulations directly utilize mesh-type phantoms. The simulation scenario involves a 10 MeV photon beam AP irradiating an adult male MRCP model. The beam, originating from a source area of 180 cm x 60 cm located at -30 cm, is directed towards the positive y-axis. The simulation outcomes are the external radiation equivalent dose conversion coefficients for various organs.

The bar graph in the figure 9 represents the equivalent doses in various organs per fluence of 10 MeV photons under AP irradiation using different simulation methods. Given that all four GPU simulation configurations are distinct modalities of our GPU MC program, and can be selected based on specific hardware conditions and simulation requirements, it is imperative to ensure the accuracy of these four computational results. We exhibit the maximum relative deviations in organ doses calculated by the four GPU methods compared to the CPU benchmark for each organ. The accompanying line graph displays the maximum deviation. Generally, the results for the most sensitive organs are consistent across all five calculation methods, with discrepancies within 5% of the benchmark, thus confirming the accuracy of the GPU program's calculations.

To further analyse the variations among different GPU simulation methods, particularly the effects of various optimization techniques, figure 10 employs a box-and-whisker plot to illustrate the distribution of dose deviations for all organs in a mesh-type phantom across different GPU calculation methods compared to the benchmark. Each box (column) in figure 10 represents the distribution of dose simulation results for a particular GPU simulation scenario, with each point indicating the relative deviation of each organ's results from the CPU benchmark. The narrower the box and closer its range to zero, the more accurate the overall simulation. Notably, to better demonstrate the impact of variance reduction optimization methods, the analysis includes simulations using only the event-based method with $10^7$ particles.

The left two columns of the figure 10, which represent simulations utilizing with $10^8$ particles, show that the relative deviations in dose calculations for most organs are within 3%. This consistency arises because the history-based and event-based methods do not fundamentally differ in their physical models, but rather in their transport mechanisms. A comparison of these two $10^8$ particle simulations (left two box plots) with the three $10^7$ particle simulations (right three box plots) demonstrates that simulations with $10^8$ particles exhibit noticeably smaller relative deviations. This observation aligns with statistical principles, where a higher particle count leads to lower statistical errors and more precise calculations.

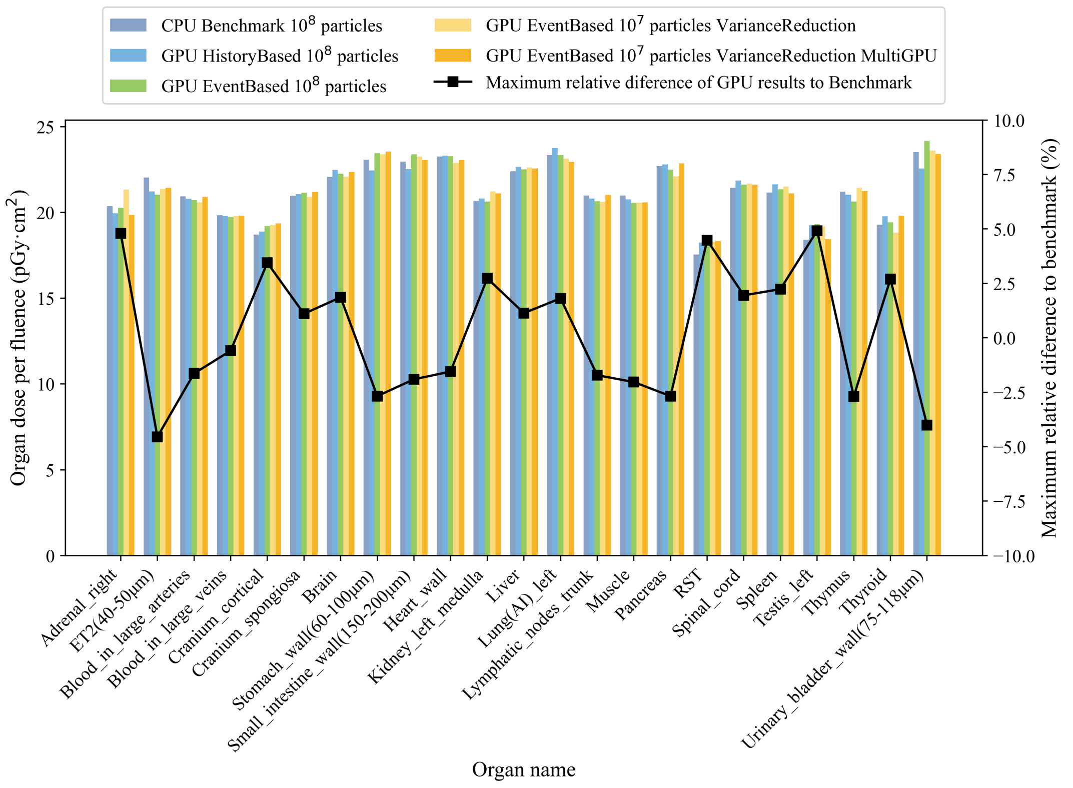


Fig. 9. (Color online) Comparison of organ equivalent doses per fluence of 10 MeV photon under AP irradiation across different simulation configurations. The bar charts represent the dose calculation results for each simulation configuration, while the line graph shows the maximum deviation of GPU-calculated results for each organ compared to the CPU benchmark.

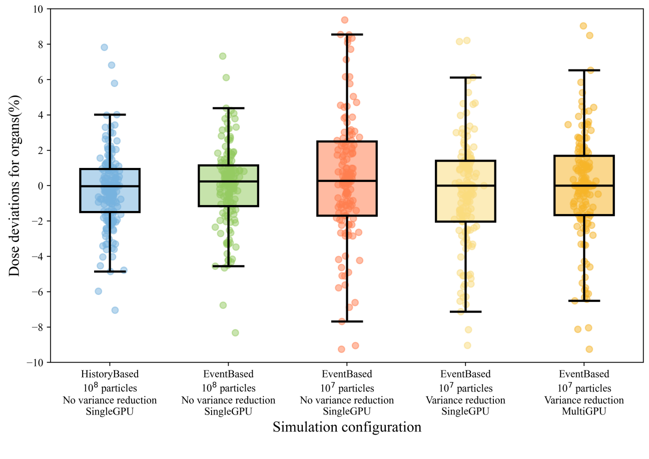


Fig. 10. (Color online) Distribution of dose deviations for all organs of the mesh-type phantom across different GPU simulation configurations compared to the benchmark results. The box represents the interquartile range (from the first quartile to the third quartile). The median line inside the box indicates the middle value of the data. The whiskers extend from the box to show the full range of the data, excluding some outliers.

Subsequent comparisons of three GPU simulation methods, each running with $10^7$ particles (the right three box plots), indicate that the range of relative dose differences among organs narrows with the application of variance reduction techniques. The results show that applying variance reduction techniques improved the accuracy of simulations with the same particle count, demonstrating the effectiveness of the method, although the accuracy still does not reach that of simulations using $10^8$ particles. Nevertheless, these differences remain within acceptable limits, but significantly reduce the number of simulation particles.

## B. Comparison of computation speed

Table 1 summarizes the differences in computation times across various simulation configurations. The CPU benchmark time represents the computation time using a single-core for simulation. The term “Time ratio to benchmark” indicates the ratio of other simulation computation times to the CPU benchmark time.

It reveals that the GPU-based program for mesh-type models developed in this study, when operated without event-based transport or variance reduction techniques, reduces computation time by a substantial factor of 570 compared to the single-core CPU MC program. This improvement is due to the lack of acceleration structures or efficient intersection algorithms in conventional CPU MC programs such

TABLE 1. Computation times across various simulation configurations.

| Program and method | Phantom type | Hardware | Total particles simulated | Simulation time (s) | Time ratio to benchmark |
|---|---|---|---|---|---|
| CPU Single-core | Tetrahedron-type | Intel Xeon CPU E5-2660 | $1 \times 10^8$ | 193322.12 | 1.00 |
| GPU + History-based | Mesh-type | RTX 4090 | $1 \times 10^8$ | 338.15 | 571.71 |
| GPU + Event-based | Mesh-type | RTX 4090 | $1 \times 10^8$ | 96.28 | 2007.92 |
| GPU + Event-based | Mesh-type | RTX 4090 | $1 \times 10^7$ | 10.34 | 18696.53 |
| GPU + Event-based + Variance reduction | Mesh-type | RTX 4090 | $1 \times 10^7$ | 12.58 | 15367.42 |
| GPU + Event-based + Variance reduction + MultiGPU | Mesh-type | 5× RTX 4090 | $1 \times 10^7$ | 3.74 | 51690.41 |

as Geant4, whether using mesh-type models or parametric tetrahedral models. Therefore, combining GPU hardware with software optimization algorithms to accelerate mesh-type models achieves faster acceleration than the CPU with tetrahedral models, and even more so compared to the CPU with mesh-type models. Since methods involving CPUs with mesh-type models are not mainstream, they are not compared here.

Employing optimization techniques, such as the event-based transport method, further enhances performance, achieving an additional fourfold acceleration when simulating the same number of particles, which results in a total speedup of 2000 times compared to the CPU benchmark. This improvement is primarily because the event-based transport method significantly reduces thread divergence and increases hardware utilization. However, it should be noted that the need for repetitive kernel function calls and additional operations like particle sorting may slightly decrease computational efficiency.

Furthermore, utilizing both event-based transport and variance reduction techniques with fewer particles decreases computation time by 18,000 times compared to the baseline, while the relative differences in organ doses remain within acceptable ranges. However, the implementation of variance reduction techniques, which includes operations such as particle splitting and roulette, can lead to an increase in computation time.

Further acceleration can be achieved by utilizing multiple GPUs in parallel. This approach involves synchronizing data across different devices. However, due to the varying operational conditions of each device, there can be discrepancies in runtime, which means that the overall simulation computation time does not decrease proportionally with the number of GPUs used.

In summary, the GPU-based MC program, which incorporates various acceleration and optimization methods, can significantly shorten the computation times for human phantom dose calculations to the order of second, achieving up to a 50,000-fold reduction compared to the single-core CPU MC program.

## IV. CONCLUSIONS

This study presents the first instance of a GPU-accelerated MC program that performs dose calculations directly using mesh-type models without requiring conversion processes such as tetrahedralization or voxelization. By directly processing polygonal models for dose calculations, the program leverages their flexible deformation capabilities and high resolution, yielding more accurate dosimetric outcomes. Additionally, this approach eliminates the need for tetrahedral conversion and mesh repair associated with tetrahedral mesh-type phantoms, thereby simplifying the computational workflow. By incorporating GPU acceleration for MC transport, constructing acceleration structures, enabling single-step transport based on particle type, and applying variance reduction techniques along with multi-GPU parallel optimization, the simulation time for a single phantom reduces from hours to seconds, achieving up to a 50,000-fold reduction compared to the single-core CPU MC program. The GPU-accelerated calculation method for mesh-type phantoms, combined with human posture capture and deformation technologies, enables real-time human dose calculations, facilitating more accurate applications in fields with stringent time constraints, such as radiotherapy and accident dose reconstruction.

There is further potential for optimization in the program. For instance, it currently transports particles based on their type. Future implementations that transport based on different reaction types may significantly reduce thread divergence. Additionally, the geometric importance for variance reduction, currently adjusted manually based on each run's outcomes, will be automated in future settings.

[1] J.H. Jeong, S. Cho, C. Lee et al, Development of Deformable Computational Model for Korean Adult Male Based on Polygon and NURBS Surfaces. Nuclear Technology **168**, (2009). doi:10.13182/nt09-a9130

[2] C.H. Kim, Y.S. Yeom, N. Petoussi-Henss et al, ICRP Publication 145: Adult Mesh-Type Reference Computational Phantoms. Ann. ICRP **49**, (2020). doi:10.1177/0146645319893605

[3] Y.S. Yeom, C. Choi, H. Han et al, Dose coefficients of mesh-type ICRP reference computational phantoms for external exposures of neutrons, protons, and helium ions. Nucl. Eng. Tech-

nol. **52**, (2020). doi:10.1016/j.net.2019.12.020

[4] C. Choi, T.T. Nguyen, Y.S. Yeom et al, Mesh-type reference Korean phantoms (MRKPs) for adult male and female for use in radiation protection dosimetry. Phys. Med. Biol. **64**, 085020 (2019). doi:10.1088/1361-6560/ab0b59

[5] C. Cumur, T. Fujibuchi, K. Hamada, Dose estimation for cone-beam computed tomography in image-guided radiation therapy using mesh-type reference computational phantoms and assuming head and neck cancer. J. Radiol. Prot. **42**, 021533 (2022). doi:10.1088/1361-6498/ac7914

[6] T.E. Kwon, H.G. Han, C. Choi et al, Enhanced internal dosimetry for alimentary tract organs in nuclear medicine based on the ICRP mesh-type reference phantoms. Radiat. Phys. Chem. **224**, 112009 (2024). doi:10.1016/j.radphyschem.2024.112009

[7] S. Moon, H. Han, C. Choi et al, Towards accurate dose assessment for emergency industrial radiography source retrieval operations: A preliminary study of 4D Monte Carlo dose calculations. Nucl. Eng. Technol. **56**, (2024). doi:10.1016/j.net.2024.09.004

[8] X. Wang, J.L. Li, Z. Wu et al, CMGC: a CAD to Monte Carlo geometry conversion code. Nucl. Sci. Tech. **31**, 82 (2020). doi:10.1007/s41365-020-00793-8

[9] S. Agostinelli, J. Allison, K. Amako et al, GEANT4-a simulation toolkit. Nucl. Instrum. Methods Phys. Res. Sect. A-Accel. Spectrom. Dect. Assoc. Equip. **506**, (2003). doi:10.1016/s0168-9002(03)01368-8

[10] Z.F. Luo, R. Qiu, M. Li et al, Study of a GPU-based parallel computing method for the Monte Carlo program. Nucl. Sci. Tech. **25**, S010501 (2014). doi:10.13538/j.1001-8042/nst.25.S010501

[11] C.H. Kim, J.H. Jeong, W.E. Bolch et al, A polygon-surface reference Korean male phantom (PSRK-Man) and its direct implementation in Geant4 Monte Carlo simulation. Phys. Med. Biol. **56**, (2011). doi:10.1088/0031-9155/56/10/016

[12] H. Si, TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator. Acm Transactions on Mathematical Software **41**, 11 (2015). doi:10.1145/2629697

[13] G. Hansen, S. Owen, Mesh generation technology for nuclear reactor simulation; barriers and opportunities. Nucl. Eng. Des. **238**, (2008). doi:10.1016/j.nucengdes.2008.05.016

[14] C.M. Poole, I. Cornelius, J.V. Trapp et al, Fast Tessellated Solid Navigation in GEANT4. IEEE Trans. Nucl. Sci. **59**, (2012). doi:10.1109/tns.2012.2197415

[15] Y.S. Yeom, J.H. Jeong, M.C. Han et al, Tetrahedral-mesh-based computational human phantom for fast Monte Carlo dose calculations. Phys. Med. Biol. **59**, (2014). doi:10.1088/0031-9155/59/12/3173

[16] H. Han, J. Kim, S. Moon et al, MPPD: A User-Friendly Posture Deformation Program for Mesh-Type Computational Phantoms. Health Phys. (2024). doi:10.1097/hp.0000000000001884

[17] Z. Peng, Y. Lu, Y. Xu et al, Development of a GPU-accelerated Monte Carlo dose calculation module for nuclear medicine, ARCHER-NM: demonstration for a PET/CT imaging procedure. Phys. Med. Biol. **67**, 06NT02 (2022). doi:10.1088/1361-6560/ac58dd

[18] L. Su, Y.M. Yang, B. Bednarz et al, ARCHERRT - A GPU-based and photon-electron coupled Monte Carlo dose computing engine for radiation therapy: Software development and application to helical tomotherapy. Med. Phys. **41**, 071709 (2014). doi:10.1118/1.4884229

[19] Q. Gong, R.I. Stoian, D.S. Coccarelli et al, Rapid simulation of X-ray transmission imaging for baggage inspection via GPU-based ray-tracing. Nucl. Instrum. Methods Phys. Res. Sect. B-Beam Interact. Mater. Atoms **415**, (2018). doi:10.1016/j.nimb.2017.09.035

[20] F.P. Vidal, P.F. Villard, Development and validation of real-time simulation of X-ray imaging with respiratory motion. Comput. Med. Imaging Graph. **49**, (2016). doi:10.1016/j.compmedimag.2015.12.002

[21] G.G. Xu, C.Z. Dong, T. Zhao et al, Acceleration of shooting and bouncing ray method based on OptiX and normal vectors correction. PLoS One **16**, e0253743 (2021). doi:10.1371/journal.pone.0253743

[22] R. Yan, L.B. Huang, H. Guo et al, RT Engine: An Efficient Hardware Architecture for Ray Tracing. Appl. Sci.-Basel **12**, 9599 (2022). doi:10.3390/app12199599

[23] A.K. Hu, R. Qiu, H. Liu et al, THUBrachy: fast Monte Carlo dose calculation tool accelerated by heterogeneous hardware for high-dose-rate brachytherapy. Nucl. Sci. Tech. **32**, 32 (2021). doi:10.1007/s41365-021-00866-2

[24] W.G. Li, C. Chang, Y. Qin et al, GPU-based cross-platform Monte Carlo proton dose calculation engine in the framework of Taichi. Nucl. Sci. Tech. **34**, 77 (2023). doi:10.1007/s41365-023-01218-y

[25] G. Franciosini, G. Battistoni, A. Cerqua et al, GPU-accelerated Monte Carlo simulation of electron and photon interactions for radiotherapy applications. Phys. Med. Biol. **68**, 044001 (2023). doi:10.1088/1361-6560/aca1f2

[26] X. Jia, X.J. Gu, J. Sempau et al, Development of a GPU-based Monte Carlo dose calculation code for coupled electron-photon transport. Phys. Med. Biol. **55**, (2010). doi:10.1088/0031-9155/55/11/006

[27] Y. Lai, Y. Zhong, A. Chalise et al, gPET: a GPU-based, accurate and efficient Monte Carlo simulation tool for PET. Phys. Med. Biol. **64**, 245002 (2019). doi:10.1088/1361-6560/ab5610

[28] Y. Li, W. Sun, H. Liu et al, Development of a GPU-superposition Monte Carlo code for fast dose calculation in magnetic fields. Phys. Med. Biol. **67**, 125002 (2022). doi:10.1088/1361-6560/ac7194

[29] S. Sharma, A. Kapadia, W. Fu et al, A real-time Monte Carlo tool for individualized dose estimations in clinical CT. Phys. Med. Biol. **64**, 215020 (2019). doi:10.1088/1361-6560/ab467f

[30] P.K. Romano, A.R. Siegel, Limits on the efficiency of event-based algorithms for Monte Carlo neutron transport. Nucl. Eng. Technol. **49**, (2017). doi:10.1016/j.net.2017.06.006

[31] K. Yang, B.S. He, Q.O. Luo et al, Stack-Based Parallel Recursion on Graphics Processors. Acm Sigplan Notices **44**, (2009). doi:10.1145/1594835.1504224

[32] S.P. Hamilton, S.R. Slattery, T.M. Evans, Multigroup Monte Carlo on GPUs: Comparison of history- and event-based algorithms. Ann. Nucl. Energy **113**, (2018). doi:10.1016/j.anucene.2017.11.032

[33] Q.M. He, Z.P. Huang, L.Z. Cao et al, The methods of CADIS-NEE and CADIS-DXTRAN in NECP-MCX and their applications. Nucl. Eng. Technol. **56**, (2024). doi:10.1016/j.net.2024.02.036

[34] L. Deng, G. Li, T. Ye et al, MCDB Monte Carlo Code with Fast Track Technique and Mesh Tally Matrix for BNCT. Journal of Nuclear Science and Technology **44**, (2007). doi:10.1080/18811248.2007.9711401

[35] A. Aman, S. Demirci, U. Güdükbay et al, Multi-level tetrahedralization-based accelerator for ray-tracing animated scenes. Computer Animation and Virtual Worlds **32**, e2024 (2021). doi:10.1002/cav.2024

[36] Z.P. Chen, J. Song, H.Q. Zheng et al, Optimal Spatial Subdivi-

sion method for improving geometry navigation performance in Monte Carlo particle transport simulation. Ann. Nucl. Energy **76**, (2015). doi:10.1016/j.anucene.2014.10.028

[37] X. Wang, Y. Yu, X. Li et al, Development of a hybrid parallelism Monte Carlo transport middleware on mesh geometry. Ann. Nucl. Energy **190**, (2023). doi:https://doi.org/10.1016/j.anucene.2023.109872

[38] M.C. Han, J.M. Seo, S.H. Lee et al, Continuously Deforming 4D Voxel Phantom for Realistic Representation of Respiratory Motion in Monte Carlo Dose Calculation. IEEE Trans. Nucl. Sci. **63**, (2016). doi:10.1109/tns.2016.2601080

[39] M. Vinkler, V. Havran, J. Sochor, Visibility driven BVH build up algorithm for ray tracing. Computers & Graphics-Uk **36**, (2012). doi:10.1016/j.cag.2012.02.013

[40] J.D. MacDonald, K.S. Booth, Heuristics for ray tracing using space subdivision. Vis. Comput. (West Germany) **6**, (1990). doi:10.1007/bf01911006

[41] W. Boukaram, G. Turkiyyah, D. Keyes, Hierarchical Matrix Operations on GPUs: Matrix-Vector Multiplication and Compression. Acm Transactions on Mathematical Software **45**, 3 (2019). doi:10.1145/3232850

[42] J.K. Boahen, A.S.G. Khalil, M.A. Hassan et al, EJUSTCO: Monte Carlo radiation transport code hybrid with ANN model for gamma-ray shielding simulation. Nucl. Sci. Tech. **34**, 144 (2023). doi:10.1007/s41365-023-01297-x

[43] S.C. Yan, R. Qiu, Z. Wu et al, Individualized dose calculation for internal exposure on radionuclide intake: GPU acceleration approach. Phys. Med. Biol. **69**, 175002 (2024). doi:10.1088/1361-6560/ad69fa

[44] Y. Lee, R. Avizienis, A. Bishara et al, Exploring the Trade-offs between Programmability and Efficiency in Data-Parallel Accelerators. ACM Trans. Comput. Syst. **31**, 6 (2013). doi:10.1145/2491464

[45] S.P. Hamilton, T.M. Evans, Continuous-energy Monte Carlo neutron transport on GPUs in the Shift code. Ann. Nucl. Energy **128**, (2019). doi:10.1016/j.anucene.2019.01.012

[46] R.M. Bergmann, J.L. Vujic, Algorithmic choices in WARP - A framework for continuous energy Monte Carlo neutron transport in general 3D geometries on GPUs. Ann. Nucl. Energy **77**, (2015). doi:10.1016/j.anucene.2014.10.039

[47] F.B. Brown, W.R. Martin, Monte Carlo methods for radiation transport analysis on vector computers. Progress in Nuclear Energy **14**, (1984). doi:https://doi.org/10.1016/0149-1970(84)90024-6

[48] A. Dubi, General statistical model for geometrical splitting in Monte Carlo. I. Transp. Theory Stat. Phys. (USA) **14**, (1985). doi:10.1080/00411458508211675

[49] A. Davis, A. Turner, Comparison of global variance reduction techniques for Monte Carlo radiation transport simulations of ITER. Fusion Eng. Des. **86**, (2011). doi:10.1016/j.fusengdes.2011.01.059

[50] H. Lee, J. Shin, J.M. Verburg et al, MOQUI: an open-source GPU-based Monte Carlo code for proton dose calculation with efficient data structure. Phys. Med. Biol. **67**, 174001 (2022). doi:10.1088/1361-6560/ac8716

[51] B.R.B. Walters, I. Kawrakow, D.W.O. Rogers, History by history statistical estimators in the BEAM code system. Med. Phys. **29**, (2002). doi:10.1118/1.1517611